Ferramenta para Geração de Código-fonte Arduino a partir de Modelos em Fritzing

Matheus Guilherme Rohde[1], Reiner Franthesco Perozzo[1]

1Curso de Sistemas de Informação – Centro Universitário Franciscano -
Caixa Postal 97010-032– Santa Maria – RS – Brasil

rohde1392@gmail.com, reiner.perozzo@unifra.br

Abstract. *This paper proposes a software tool able to generate source code for the Arduino electronic platform. Among the main features of the proposal tool, include: (i) the possibility of linking actions to the devices of a model created in Fritzing tool, (ii) the attempt to reduce the complexity of application development for the Arduino platform and (iii) the ability to generate code automatically for a particular model designed in Fritzing. At the end, the work was validated using the tool in a prototype also developed in the proposal.*

Resumo. *Este trabalho apresenta a proposta de criação de uma ferramenta capaz de gerar código-fonte para a plataforma de prototipagem eletrônica Arduino. Dentre as principais características da ferramenta proposta, destacam-se: (i) a possibilidade de associar ações aos dispositivos de um modelo criado na ferramenta Fritzing, (ii) a tentativa de redução da complexidade do desenvolvimento de aplicações para a plataforma Arduino e (iii) a capacidade de gerar código, automaticamente, para um determinado modelo projetado no Fritzing. Ao final, o trabalho foi validado com a utilização da ferramenta em um protótipo desenvolvido também no âmbito da proposta.*

1. Introdução

Ao longo dos últimos anos vem sendo observada uma crescente demanda por sistemas microcontrolados, projetados para atenderem as mais diversas áreas. Dentre as quais, é possível destacar a automação industrial, automação predial/residencial, setor automobilístico, telecomunicações, dentre outras [Atmel 2016]. Por exemplo, especificamente na área de automação residencial, tais sistemas podem atuar como sensores de luminosidade, climatização, segurança, controle de acesso, etc. O microcontrolador é um dispositivo que reúne, em um único circuito integrado, diversos componentes de um sistema computacional, podendo ser programável para aplicações diversas. Os mais conhecidos no mercado, atualmente, são o PIC – fabricado pela Microchip e o Atmega fabricado pela Atmel [Corteletti 2006].

Os microcontroladores, normalmente, compõem um sistema embarcado (contemplando sua integração com sensores, atuadores e, também, com outros microcontroladores) [Gallassi e Martins 2011]. Tais projetos de sistemas embarcados podem ser prototipados com o auxílio de ferramentas capazes de modelar os dispositivos de um cenário microcontrolado, juntamente com as suas conexões. Um exemplo dessas

ferramentas é o Fritzing, um *software* que se destaca pela possibilidade de criação de diagramas eletrônicos. Nele, é possível obter a geração automática do desenho em um esquema elétrico e até em um *layout* de *Printed Circuit Board* (PCB), o que permite imprimir parte do circuito eletrônico a ser construído [Fritzing 2016].

No que tange os microcontroladores e a prototipagem de sistemas embarcados, há uma plataforma que vem obtendo destaque: o Arduino [Arduino 2016], o qual consiste numa plataforma de prototipagem eletrônica de código aberto, baseado numa placa microcontroladora e um *Integrated Development Environment* (IDE). A partir disso, inúmeros sistemas eletrônicos microcontrolados tem surgido. No entanto, o desenvolvimento de aplicações baseadas em sistemas microcontrolados podem enfrentar dificuldades acerca dos métodos de programação utilizados. Desse modo, a existência de ferramentas que auxiliem na construção de código ou até mesmo que gerem código-fonte automaticamente torna-se uma solução para os programadores e usuários que não saibam, necessariamente, programar.

Além disso, com a demanda no desenvolvimento de sistemas microcontrolados, podem ser necessários *softwares* que auxiliem na geração de código-fonte, cuja finalidade é otimizar o tempo e facilitar o acesso de usuários ao desenvolvimento dessas aplicações. Desse modo, o presente trabalho propõe uma ferramenta capaz de gerar código fonte, para a plataforma de prototipagem eletrônica Arduino, através de modelos produzidos no *software* Fritzing. Tal ferramenta permitirá, além da criação estrutural, também, a associação comportamental aos dispositivos, em que comandos pré-estabelecidos para alguns dispositivos estarão disponíveis na ferramenta proposta.

1.1. Objetivo Geral

Desenvolver uma ferramenta capaz de gerar código-fonte para a plataforma Arduino (englobando estrutura de código e comportamento) a partir da importação de modelos criados pela ferramenta Fritzing.

1.2. Objetivos Específicos

- Importar os modelos da ferramenta Fritzing;

- Extrair informações estruturais do cenário criado no Fritzing;

- Associar comportamento aos modelos estruturais originalmente obtidos;

- Gerar código-fonte para o Arduino, contemplando a estrutura e comportamento para o cenário microcontrolado projetado no Fritzing.

2. Revisão Bibliográfica

Nesta seção são apresentados alguns conceitos e tecnologias relacionados com a elaboração deste trabalho. São abordados, a ferramenta Fritzing, a linguagem de marcação *eXtensible Markup Language* (XML), os microcontroladores (com ênfase na plataforma de prototipagem eletrônica Arduino), a linguagem de programação Java e a linguagem de consulta *XML Path Language* (XPath).

2.1. Fritzing

Fritzing é uma ferramenta de *software open-source* multiplataforma, que foi desenvolvida na Universidade de Ciências Aplicadas de Postdam, na Alemanha. Esse *software* permite a criação de esquemas e diagramas eletrônicos, prototipagem e *layout* de placas de PCB, usado em placas Arduino, Raspberry pi e Beaglebone [Fritzing 2016].

A ferramenta está na versão 0.9.3 e possui diversas funcionalidades, sendo possível (dentre outros recursos) desenhar o circuito em uma *protoboard*. No módulo de esquemático tem-se a conversão da *protoboard* em símbolos eletrônicos correspondentes. A seguir existe o modo PCB que converte o modelo projetado em uma imagem de circuito impresso. Finalmente, há um modo código-fonte em que é possível escrever ou importar um código-fonte, não necessitando da utilização do *software* Arduino IDE. A ferramenta possui inúmeros componentes que podem ser configurados e editados, além de oferecer suporte a vários formatos para exportação [Fritzing 2016]. A Figura 1 ilustra a ferramenta Fritzing de modo geral.

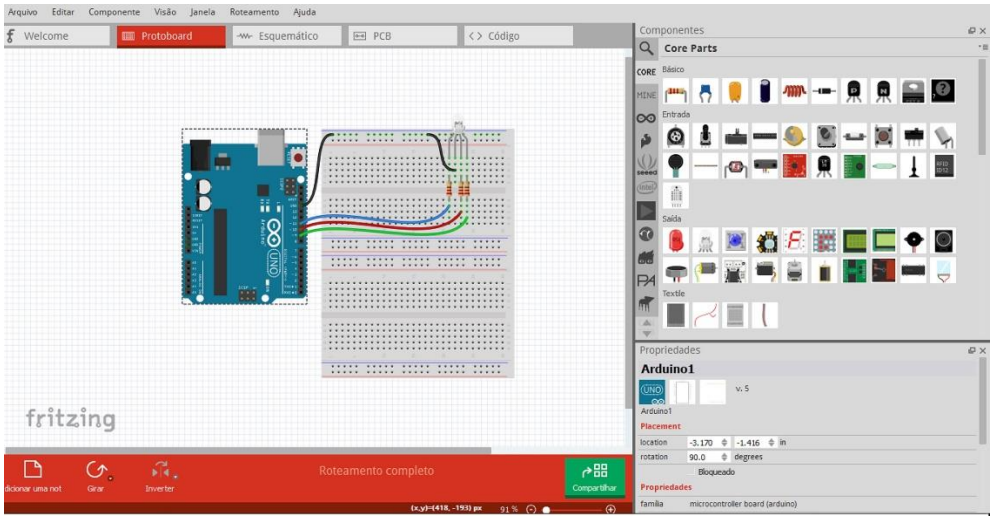

Figura 1 – Interface do software Fritzing.

2.2. XML

A linguagem de marcação XML foi desenvolvida com o apoio do *World Wide Web Consortium* (W3C), com o propósito de que os programadores pudessem criar suas próprias *tags* nas páginas *web*. Ela é uma linguagem de marcação descritiva extensível, assim contendo características como: independência a plataformas de *software* e de *hardware*, buscando minimizar os impactos de manutenção de código e numa tentativa de facilitar a reutilização de código em detrimento do *HyperText Markup Language* (HTML) que só pode expandir através de pós-processamentos característicos e não através de novos elementos adicionados à linguagem [Ramalho e Henriques 2002].

O XML possui algumas funcionalidades importantes, dentre elas a extensibilidade que permite ao usuário definir novas marcações, acrescentar atributos e relacionar com a estrutura de código já existente. A estrutura das classes é definida pelo usuário, que também

pode validar a estrutura do código-fonte através do *Document Type Definition* (DTD) [Ramalho e Henriques 2002].

2.3. Microcontroladores

Os microcontroladores, historicamente, possuem um papel relevante nas mais diversas áreas [Atmel 2016a], [Atmel 2016b], pois permitem a automatização e o processamento para a tomada de decisão existente em diversos dispositivos. Por serem pequenos e de baixo custo - se comparados com os *Personal Computers* (PCs) tradicionais - são uma escolha interessante para controlar dispositivos digitalmente.

Um microcontrolador pode ser definido como um computador em um *chip*, contendo processador, acesso a memória e periféricos de entrada e saída. A arquitetura de um microcontrolador consiste em núcleo de processamento, barramento e periféricos [Oki e Mantovani 2013].

Os primeiros microcontroladores desenvolvidos foram os AVR pela ATMEL, com a arquitetura desenvolvida na linguagem C e um núcleo de 8 *bits*. Existem vários tipos de microcontroladores disponíveis atualmente, os quais podem ser utilizados em diversas soluções embarcadas [Oki e Mantovani 2013]. Um exemplo, pode ser obtido com a plataforma Arduino, a qual é voltada para a prototipagem eletrônica. Seu *hardware* é composto por uma placa de prototipagem na qual são construídos os projetos, sendo que o seu *software* é uma IDE baseada na linguagem de programação C e C++, onde são criados os *sketch* (aplicações Arduino). Logo após, é feito o *upload* dos *sketch* por comunicação serial para a placa de prototipagem Arduino que o executa montando a aplicação [Embarcados 2016].

2.4. Java

A linguagem computacional Java foi desenvolvida no ano de 1995 pela Sun Microsystems. Essa linguagem de programação é orientada a objetos, isso significa que é possível reutilizar e realizar a manutenção do código-fonte de forma mais acessível se comparada com as linguagens estruturadas clássicas [Indrusiak 1996]. Essa linguagem de programação possui bibliotecas de classes Java, também conhecidas como API. São diversas coleções de classes prontas que os programadores utilizam numa tentativa de reduzir tempo no processo programação [Deitel e Deitel 2005].

O ambiente de desenvolvimento Java possui cinco fases: a primeira fase é escrever o programa e armazenar no disco em um arquivo de extensão .java. A seguir o compilador cria *bytecodes* que são armazenados no disco com a extensão *.class*. Na próxima fase o carregador de classe lê os arquivos *.class* que contêm os *bytecodes,* sendo que em seguida é disponibilizado em memória pelo disco. Logo após, o verificador de *bytecodes* realiza a confirmação que todos *bytecodes* são válidos e não desrespeitam as restrições de segurança do Java. Assim, a máquina virtual dessa linguagem de programação - o *Java Virtual Machine* (JVM) - executa o programa lendo os *bytecodes* e traduzindo-os para uma linguagem em que o computador possa entender [Deitel e Deitel 2005].

2.5. XPath

Considerado o principal elemento no padrão *World Wide Web Consortium* (WC3) *EXtensible Stylesheet Language Transforming* (XSLT) no qual consiste em transformar um arquivo XML em um tipo de documento que possa ser reconhecido pelo browser, como o HTML por exemplo, o *XML Path Language* (XPath) é uma linguagem de consulta que possui um conjunto de regras de sintaxe, sendo utilizada para análise e extração de dados de documentos XML.

O XPath opera na estrutura abstrata, lógica de um documento XML permitindo a manipulação de *strings*, números e *booleanos*, através de uma biblioteca de funções e utilizando expressões de caminho, como a de sistemas de arquivos em um sistema operacional, para a localização de elementos do mesmo [XPath 2016].

3. Trabalhos relacionados

Esta subseção apresenta a utilização de algumas das abordagens existentes no domínio deste trabalho. Por meio de uma revisão de trabalhos correlatos, são apresentadas algumas soluções para geração de códigos-fonte que utilizam diferentes linguagens de programação, com variadas estruturas de arquivos e, sintaticamente, distintos.

3.1. Gerador de código Java baseado em modelos de dados

A necessidade de otimizar a geração de códigos fez com que determinados *frameworks* se destacassem por tornar os processos mais ágeis visando a manutenção de qualidade, produtividade, consistência do código e abstração, o que permite o uso de diferentes plataformas.

O estudo realizado por [Ribeiro, Novais e Mendes 2005] fundamentou-se na criação de uma ferramenta para geração de código, na qual se comunica com dois sistemas gerenciadores de bancos de dados, o MySQL e o PostGreSQL. A ferramenta desenvolvida na linguagem Java e integrada ao IDE Eclipse - sob forma de *plug-in* - realiza a extração de dados do banco através de APIs, gerando código-fonte para linguagem Java. Diante disso, a ferramenta gera as classes para acesso e manipulação do banco de dados e também classes que implementam a interface com o usuário.

A ferramenta foi validada em um modelo de dados, no qual baseia-se na geração automática das telas de inclusão, de alteração e de exclusão. O resultado obtido mostrou que a ferramenta foi capaz de gerar as principais telas do modelo de dados. No entanto, algumas desvantagens foram identificadas, tais como: as telas geradas possuem sempre o mesmo formato e a restrição de interfaces de comunicação com somente dois tipos de SGBDs.

3.2. Ferramenta de geração automática de código

Esse trabalho [Castro 2010] propõe a criação de duas ferramentas para geração automática de código, intituladas CodeGen e BlueBox, para otimizar a produção de código no formato do *framework* Iguassu.

Como entrada de dados, ambas ferramentas utilizam o arquivo *XML Metadata Interchange* (XMI), para obter informações do modelo *Unified Modeling Language* (UML). Na ferramenta BlueBox os dados são processados conforme as regras contidas nos *templates* da *engine* Velocyti e, dessa forma, o código-fonte é gerado. A outra ferramenta denominada CodeGen realiza a geração de código-fonte para cada classe do *framework* Iguassu. Para isso, a folha de estilo *eXtensible Stylesheet Language Transformations* (XSLT) é utilizada como *template* para a geração de código.

O BlueBox se mostrou mais eficiente que o CodeGen pelo fato de realizar um pré-processamento das informações antes de enviá-las para os *templates*. Assim, os *templates* tornam-se mais legíveis do que as folhas de estilos XSL, facilitando tanto a manutenção como a aprendizagem da programação.

3.3. Criação de um Framework para integração entre o Sistema Brasileiro de TV Digital Terrestre e Ambientes Inteligentes

Esse trabalho [Perozzo 2011] propõe a criação de uma ferramenta com a finalidade de integrar ambientes inteligentes (que são cenários automatizados cientes do contexto) com o sistema brasileiro de TV digital. Trata-se de um *framework* capaz de gerar automaticamente um código totalmente orientado a objetos, o que permite a criação de cenários de automação independentes, que visa a autonomia do usuário já que esse torna-se capaz de gerenciar qualquer dispositivo de automação em seu ambiente.

Nesse trabalho é detalhado, ainda, as fases para a construção das aplicações interativas: a ferramenta construída possui uma lista de dispositivos de automação presentes no ambiente inteligente. Então, a partir dessa lista, são criados os serviços. Assim, um cenário pode conter vários serviços nos quais possuem vários dispositivos. Esse trabalho propõe um alto nível de abstração em automação residencial, permitindo que pessoas que não conheçam tais tecnologias sejam capazes de criar, a partir da ferramenta, aplicações interativas em seu ambiente.

3.4. Considerações sobre os trabalhos relacionados

Os estudos apresentados para realização do presente trabalho possuem uma ideia em comum: o desenvolvimento de uma ferramenta que gere código, automaticamente, para uma linguagem de programação. Cada um com suas tecnologias, linguagens de programação e objetivos específicos. Por outro lado, este trabalho se diferencia dos demais pelo fato dele ter a capacidade de gerar código para plataforma Arduino a partir de modelos projetados pela ferramenta Fritzing. Um dos grandes diferenciais deste trabalho é, além de código estrutural, a ferramenta consegue gerar código comportamental para uma aplicação baseada em microcontroladores.

4. Proposta

O desenvolvimento de sistemas microcontrolados são processos nos quais os desenvolvedores necessitam possuir o conhecimento de linguagens de programação e de plataformas de prototipagem eletrônica, além do tempo para implementar soluções com todas essas tecnologias. Com isso, utilizar ferramentas que possam auxiliar nesse processo,

possibilitaria a otimização e ganho de tempo no desenvolvimento do código-fonte dos modelos eletrônicos para os programadores ou por quem não conhece programação mas deseja implementar sistemas microcontrolados.

Diante desse contexto, o presente trabalho propõe o desenvolvimento de uma ferramenta para geração automática de código-fonte para a plataforma Arduino. Desse modo, toda a estrutura modelada e prototipada no Fritzing é convertida, pela ferramenta proposta, em código-fonte na linguagem Arduino. Além da geração estrutural do código, a ferramenta possibilita a inserção de comportamento ao cenário prototipado. O tipo do comportamento a ser executado para cada componente do cenário é dado por uma lista de funcionalidades pré-definidas na ferramenta, em que o usuário seleciona a funcionalidade desejada.

Na Figura 2 é ilustrada uma visão geral da ferramenta proposta nesse trabalho.

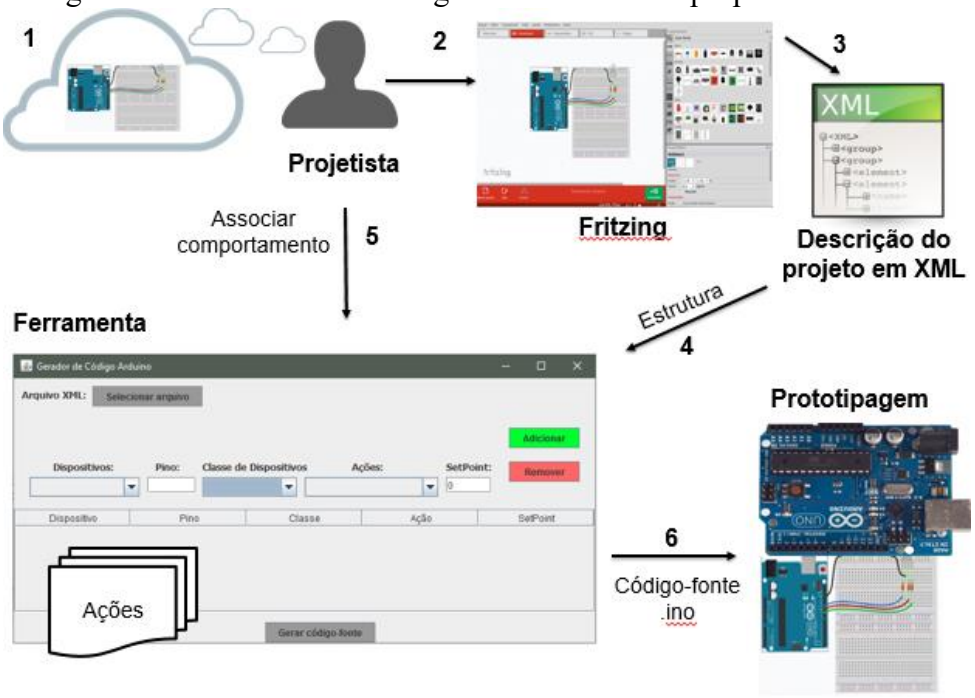

Figura 2 – Visão geral da proposta.

Conforme ilustrado na Figura 2, a ferramenta proposta segue o seguinte fluxo: o projetista deseja modelar um cenário eletrônico utilizando a ferramenta Fritzing. Assim, obtêm-se um modelo de um protótipo eletrônico Arduino, no qual é exportado para o formato de arquivo XML. Nesse arquivo está contida, detalhadamente, a estrutura do modelo projetado. Então, a ferramenta proposta deve realizar a importação e (obrigatoriamente) identificar todos dispositivos, com suas respectivas conexões presentes no cenário. Logo, o projetista será capaz de selecionar uma classe de dispositivo, que está pré-definida na ferramenta. Consequentemente, é exibida uma de lista ações para aquele tipo de classe de dispositivo selecionada permitindo que o projetista escolha qual funcionalidade o dispositivo assuma. Ainda, o projetista é capaz de adicionar (em uma tabela) os dispositivos do cenário para quais deverão ser gerados código-fonte. Por fim,

com todos os dispositivos incluídos na tabela, a ferramenta gera código-fonte para a plataforma eletrônica Arduino (extensão .ino), o qual será salvo em um arquivo, permitindo que o projetista possa obter a ação do cenário projetado inicialmente no Fritzing.

4.1. Projeto

Dada a visão geral da proposta, o presente trabalho está embasado no uso de metodologias ágeis, visando favorecer o desenvolvimento do mesmo. Diante disso, a metodologia utilizada neste trabalho é a *Feature Driven Development* (FDD), pelo motivo de atender os requisitos do domínio da proposta a ser desenvolvida [Retamal 2008].

A metodologia FDD contém cinco processos: desenvolver modelo abrangente, criar lista de funcionalidades, planejar por funcionalidade, arquitetar por funcionalidade e construir por funcionalidade. Em cada um desses cinco processos, existe uma lista de tarefas a serem executadas, possuindo os requisitos para sua execução [Retamal 2008].

a) Diagrama de classes do projeto

Conforme a metodologia utilizada para o desenvolvimento da proposta, a primeira fase da FDD apresenta o projeto da proposta, que compreende em desenvolver um modelo abrangente, construir uma lista de funcionalidades e realizar o planejamento por funcionalidade.

No primeiro processo da FDD é realizado um estudo sobre o domínio do sistema, no qual é gerado um modelo de objetos através de diagramas de classes que servem como parâmetro para uma determinada área de domínio, havendo a possiblidade de sofrer alterações durante as diversas iterações que o projeto vai ser executado, permanecendo em constante atualização até sua conclusão [Retamal 2008]. Dessa forma, a Figura 3 apresenta o diagrama de classes referente a proposta da ferramenta a ser desenvolvida no âmbito deste trabalho:

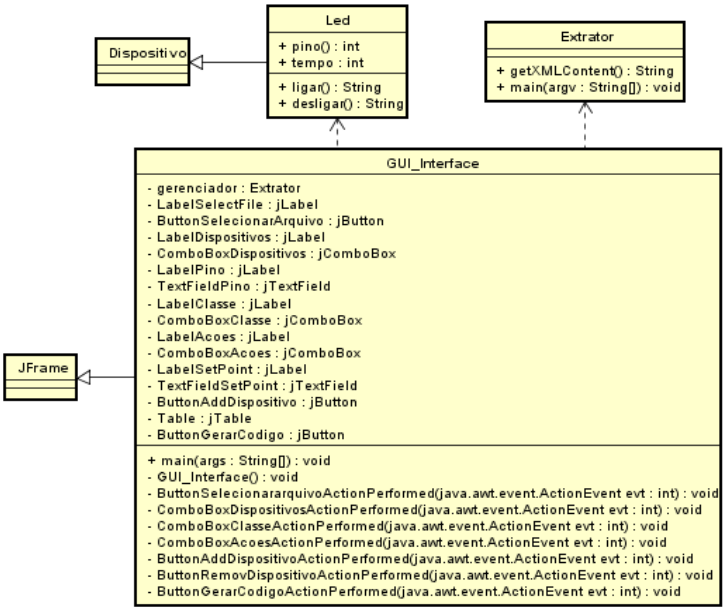

Figura 3 – Diagrama de classes da ferramenta.

O diagrama de classes, representado na Figura 3 da ferramenta desenvolvida apresenta cinco classes usadas para recebimento das informações, processamento e geração do código-fonte para a plataforma Arduino. A classe GUI_Gerenciador é herdeira da classe Frame, ou seja, ela é uma classe usada para criar a interface gráfica da ferramenta. Assim, essa classe é a principal, pois é nela que são instanciadas a classe Extrator e Led. A classe Extrator é onde são extraídas as informações necessárias dos arquivos XML gerados no Fritzing. A classe Led é herdeira de Dispositivo e define o comportamento para o tipo de dispositivo contido no modelo projetado. Assim como a classe Led construída, outras classes podem ser adicionadas na proposta, tais como Motor, Sensor e outras. Essa funcionalidade é suportada pelo uso do conceito de herança, presente na orientação a objetos. Nesse caso, novas classes de dispositivos necessárias em algum projeto específico poderiam herdar da classe Dispositivo (mapeada na proposta) e implementar novos recursos.

Seguindo a metodologia, é criada a lista de funcionalidades, a qual serve para identificar todas as funcionalidades que se encontram na ferramenta e que atendam aos requisitos [Retamal 2008], sendo elas:

1. Importar modelo Fritzing com estrutura do protótipo;
2. Listar os dispositivos da estrutura modelada;
3. Listar pinos correspondentes aos dispositivos da estrutura modelada;
4. Importar as classes de dispositivos com os seus respectivos comandos;
5. Atribuir ações aos dispositivos do modelo;
6. Definir um *setpoint* para a ação escolhida;
7. Adicionar os dispositivos em uma tabela;
8. Gerar código-fonte .ino, automaticamente.

Baseado na lista de funcionalidades, a próxima etapa da FDD estabelece a ordem na qual serão implementadas as funcionalidades com base das dependências entre elas e o grau de complexidade das funcionalidades a serem implementadas [Retamal 2008].

b) Diagrama de sequência do projeto

A Figura 4 apresenta o diagrama de sequência da ferramenta auxiliando na compreensão do funcionamento dos processos internos da ferramenta.

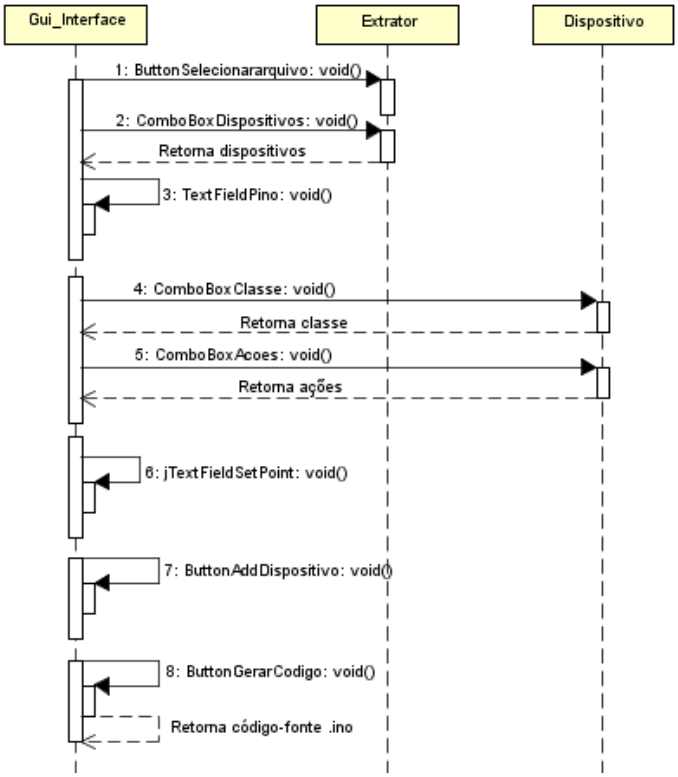

Figura 4 – Diagrama de sequência da ferramenta.

4.2. Implementação

Diante da construção das funcionalidades, destaca-se nessa etapa alguns trechos de código que são importantes quanto ao desenvolvimento da ferramenta.

a) Implementação da classe Extrator

A Figura 5 e Figura 6 representam a classe Extrator, na qual possuem o código com as *querys* construídas manualmente, responsáveis por extrair os dados necessários (dispositivos com seus respectivos pinos) do arquivo XML. A Figura 5 ilustra a *query* na linguagem de consulta XPath, que serve para filtrar somente os pinos digitais e analógicos do Arduino. A Figura 6 ilustra a outra *query* na qual extrai os nomes dos dispositivos.

```
//query para extrair os pinos digitais e analógicos do arquivo XML.
XPath xPath = XPathFactory.newInstance().newXPath();
NodeList pin = (NodeList) xPath.evaluate("//net/connector[@name=\"D0/RX\" or @name=\"D1/TX\" or "
        + "@name=\"D2\" or @name=\"D3 PWM\" or @name=\"D4\" or @name=\"D5 PWM\" or @name=\"D6 PWM\" or "
        + "@name=\"D7\" or @name=\"D8\" or @name=\"D9 PWM\" or @name=\"D10 PWM/SS\" or "
        + "@name=\"D11 PWM/MOSI\" or @name=\"D12/MISO\" or @name=\"D13/SCK\" or @name=\"A0\" or "
        + "@name=\"A1\" or @name=\"A2\" or @name=\"A3\" or @name=\"A4/SDA\" or @name=\"A5/SCL\" and "
        + "@name!=\"GND\" and @name!=\"5V\" and @name!=\"VIN\" and @name!=\"0\" and "
        + "@name!=\"Pin 0\" and @name!=\"Pin 1\" and @name!=\"Pin 2\" and @name!=\"Pin 3\" and "
        + "@name!=\"Pin 4\" and @name!=\"Pin 5\" and @name!=\"Pin 6\" and @name!=\"Pin 7\"and "
        + "@name!=\"Pin 8\" and @name!=\"pin 0\" and "
        + "@name!=\"pin 1\"]", doc.getDocumentElement(), XPathConstants.NODESET);
```

Figura 5 – *Query* para extração dos pinos do arquivo XML.

```
//query para extrair os nomes dos dispositivos do arquivo XML.
XPath xPaths = XPathFactory.newInstance().newXPath();
NodeList lab = (NodeList) xPaths.evaluate("//net/connector[@id=\"connector1\" or"
        + "@id=\"connector2\" and @id=\"connector0\"]/part[@title!=\"Arduino Uno (Rev3)\" and"
        + "@title!=\"220Ω Resistor\" and @title!=\"10kΩ Resistor\" and"
        + "@title!=\"Rectifier Diode\" and @title!=\"Relay\" and"
        + "@title!=\"Optocoupler\" and"
        + "@title!=\"NPN-Transistor\"]", doc.getDocumentElement(), XPathConstants.NODESET);
```

Figura 6 – *Query* para extração dos dispositivos do arquivo XML.

A Figura 7 representa um trecho da estrutura do arquivo XML utilizado no cenário de testes na qual a classe Extrator faz a consulta para extração de dados. Cada bloco *net* do arquivo contém as informações que correlacionam a conexão da *protoboard* com a placa Arduino de determinado dispositivo. Assim, a *query* para extração dos dispositivos consegue encontrar o nodo (*part*) contendo o atributo (*label*) de valor Led 1. Da mesma forma, a *query* para extrair os pinos realiza uma busca encontrando o nodo (*connector*) contendo o atributo (*name*) de valor D7. O valor D7, que se refere ao pino informado no arquivo não é aceito como um valor do tipo pino na linguagem Arduino. Logo, foi implementado um código para modificar, no caso, o D7 em 7 e assim sucessivamente para todos pinos analógicos e digitais, formalizando os pinos com a sintaxe correta.

```
<net>
 <connector id="connector68" name="D7">
  <part title="Arduino Uno (Rev3)" id="18471540" label="Arduino1"/>
 </connector>
 <connector id="connector1" name="anode">
  <part title="Red (633nm) LED" id="17224590" label="Led 1"/>
 </connector>
</net>
```

Figura 7 – Trecho do XML que contém as informações de pino e dispositivo.

b) Implementação da classe Interface

A classe mais importante do projeto é denominada Interface (Figura 8), a qual consiste na criação da interface com o usuário, permitindo ao mesmo, selecionar o arquivo XML (onde estão contidos os dados extraídos pela classe Extrator) (8A), realizar a associação dos dispositivos (8B) e pinos (8C) do modelo com seu tipo de classe (8D) e, também, a associação das ações permitidas para determinada classe (8E). Além disso, é possível atribuir um *setpoint* a ação selecionada (8F). Em seguida com o botão adicionar (8G) os dados são inseridos em uma tabela (8H) possibilitando a visualização para quais dados é gerado o código-fonte ao se clicar no botão gerar código-fonte (8I).

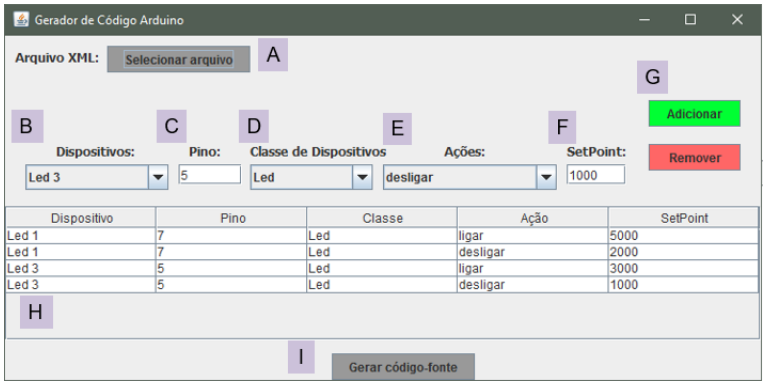

Figura 8 – Interface criada.

Por fim, essa classe possui todas as informações a serem processadas para a geração do código-fonte (extensão .ino) para a plataforma Arduino. A Figura 9 ilustra o trecho de código onde é gerado o código-fonte com os dados coletados.

```
arquivo.write("void setup(){\r\n");

ArrayList<String> contentList = new ArrayList();
int row = jTable.getRowCount();

for (int u = 0; u < row; u++) {
    for (int j = u; j < row; j++) {
        String ledPino = jTable.getValueAt(j, 1).toString();
       if( contentList.contains(ledPino)){

        } else {
          arquivo.write("pinMode(" +ledPino+ ",OUTPUT);\r\n");
          contentList.add(ledPino);
           break;
        }
}}
arquivo.write("}\r\nvoid loop(){\r\n");

for (int u = 0; u < jTable.getRowCount(); u++) {
    String ledPino = jTable.getValueAt(u, 1).toString();
    String nomeClasse = jTable.getValueAt(u, 2).toString();
    String nomeMetodo = jTable.getValueAt(u, 3).toString();
    String delay = jTable.getValueAt(u, 4).toString();
    Method method = Class.forName(nomeClasse).getMethod(nomeMetodo, Integer.TYPE, Integer.TYPE);
    arquivo.write("" + method.invoke(nomeMetodo, Integer.parseInt(ledPino), Integer.parseInt(delay)) + "");

}
arquivo.write("\r\n}");
```

Figura 9 – Trecho de código para geração de código-fonte .ino.

c) Implementação da classe Led

A Figura 10, apresenta a classe Led, na qual possui os métodos ligar e desligar para os *Light Emitting Diodes* (LED's) que podem estar contidos em cenários projetados no Fritzing. Cada método possui dois parâmetros, em que pino é o local onde o dispositivo está conectado na placa Arduino e tempo refere-se a um *delay* para o dispositivo (que encontra-se no campo *setpoint* da ferramenta). Assim, cada método implementado, ligar e desligar, retorna respectivamente o código Arduino para as ações desejadas.

```
public class Led {
    // método para ligar o Led.
    public static String ligar(int pino, int tempo) {
        return "digitalWrite(" + pino + ",HIGH);\r\n"+"delay("+ tempo +");\r\n";
    }
    // método para desligar o Led.
    public static String desligar(int pino, int tempo) {
        return "digitalWrite(" + pino + ",LOW);\r\n"+"delay("+ tempo +");\r\n";
    }
}
```

Figura 10 – Classe Led contendo os métodos ligar e desligar para os dispositivos LED's.

5. Cenário de testes e validação

Diante do contexto em que se enquadra a proposta deste trabalho, a ferramenta desenvolvida foi validada em um cenário de testes em que consiste ligar e desligar quatro LED's de maneira alternada. A justificativa de se optar por este cenário é que para outros tipos de dispositivos seria necessário a implementação de novos campos na interface da

ferramenta para suportar a adição de valores, além da inclusão de novas classes de dispositivos específicas para cada dispositivo.

Dessa forma, o cenário representado pela Figura 11 foi projetado no Fritzing e ilustra uma *protoboard* com os seguintes dispositivos: placa Arduino Uno Rev 3, fios *jumper's*, quatro LED´s e quatro resistores 220 Ohm. Desse modo, a ferramenta proposta neste trabalho deveria ser capaz de listar os componentes do modelo original, permitindo ao usuário inserir comportamentos aos LED's, gerando código-fonte automaticamente para o modelo. Assim, a Figura 11 ilustra o cenário para controle de LED's.

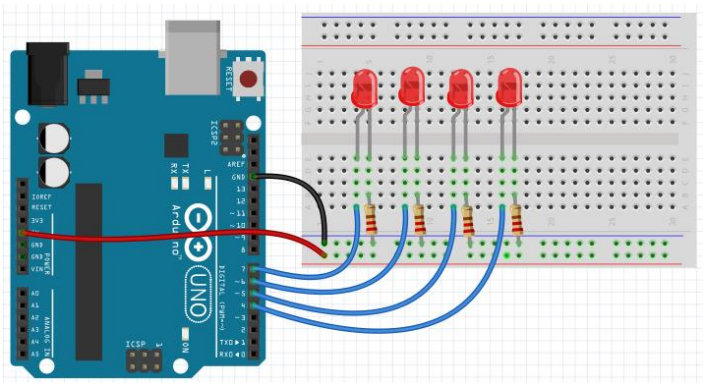

Figura 11 – Ilustração do cenário para controle de LED's.

Com o cenário construído no Fritzing (Figura 11), antes da exportação para um arquivo XML os dispositivos são renomeados no próprio Fritzing, pelo motivo dos *labels* dos dispositivos no Fritzing serem exibidos de forma padronizada, o que dificulta diferenciar um dispositivo do outro. Assim resultando na compreensão de qual dispositivo será manipulado na ferramenta. Considerando este cenário, a Figura 12 ilustra a interface da ferramenta construída bem como as anotações para o detalhamento das atividades desenvolvidas nela.

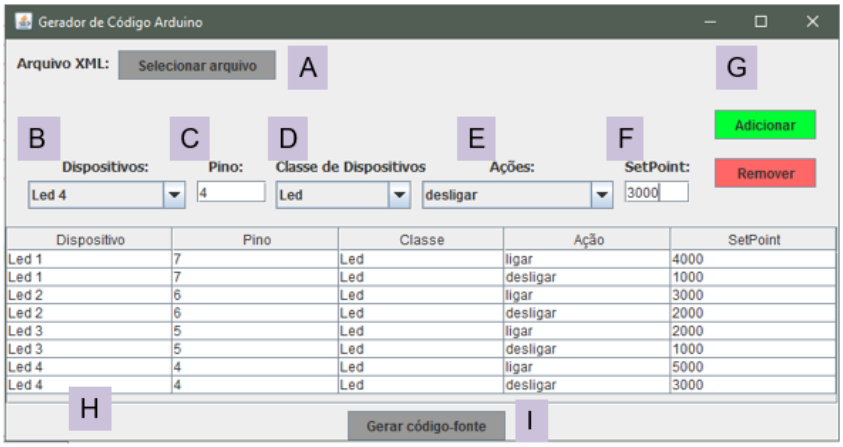

Figura 12 – Passos para utilização da ferramenta proposta.

Diante da exportação do arquivo é realizada a seleção do arquivo XML pela interface da ferramenta (12A), na qual deve listar os quatro LED's (12B) como, também os pinos em que estão conectados (12C). Em seguida, é seleciona a classe de dispositivo Led

referente ao tipo de dispositivo escolhido (12D). Na sequência, uma lista de ações é apresentada (12E), possibilitando atribuir um *setpoint* (12F). Clicando no botão Adicionar (12G), todas informações são adicionadas na tabela, para que o usuário identifique quais dispositivos contemplarão o código-fonte gerado (12H). Caso contrário, pode ser removida a informação da tabela com o botão Remover. Por último, ao clicar no botão gerar código-fonte (12I), a ferramenta gera um arquivo .ino (em um diretório do computador) contendo todo o código-fonte do protótipo, o qual pode ser imediatamente carregado e executado na IDE Arduino em que o protótipo eletrônico está conectado.

A Figura 13 ilustra a trecho principal do arquivo XML, em que estão contidas as informações necessárias para a geração de código-fonte para o cenário de LED's.

```
<net>
 <connector id="connector65" name="D4">
  <part title="Arduino Uno (Rev3)" id="18471540" label="Arduino1"/>
 </connector>
 <connector id="connector1" name="anode">
  <part title="Red (633nm) LED" id="17224620" label="Led 4"/>
 </connector>
</net>
<net>
 <connector id="connector66" name="D5 PWM">
  <part title="Arduino Uno (Rev3)" id="18471540" label="Arduino1"/>
 </connector>
 <connector id="connector1" name="anode">
  <part title="Red (633nm) LED" id="17224610" label="Led 3"/>
 </connector>
</net>
<net>
 <connector id="connector67" name="D6 PWM">
  <part title="Arduino Uno (Rev3)" id="18471540" label="Arduino1"/>
 </connector>
 <connector id="connector1" name="anode">
  <part title="Red (633nm) LED" id="17224600" label="Led 2"/>
 </connector>
</net>
<net>
 <connector id="connector68" name="D7">
  <part title="Arduino Uno (Rev3)" id="18471540" label="Arduino1"/>
 </connector>
 <connector id="connector1" name="anode">
  <part title="Red (633nm) LED" id="17224590" label="Led 1"/>
 </connector>
</net>
```

Figura 13 – Trecho do arquivo XML contendo os dados do cenário de LED's.

Considerando o cenário da Figura 11, a ferramenta foi capaz de gerar automaticamente o código-fonte na linguagem Arduino. O código gerado apresentou a sintaxe correta da linguagem Arduino, conseguindo atribuir ações de ligar (*HIGH*) e desligar (*LOW*), além de um *setpoint* (*delay*) para os LED's do modelo. A Figura 14 apresenta o código-fonte gerado pela ferramenta.

```
void setup(){
  pinMode(7, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(4, OUTPUT);
}
void loop(){
  digitalWrite(7, HIGH);
  delay(2000);
  digitalWrite(7, LOW);
  delay(1000);
  digitalWrite(6, HIGH);
  delay(3000);
  digitalWrite(6, LOW);
  delay(1000);
  digitalWrite(5, HIGH);
  delay(1000);
  digitalWrite(5, LOW);
  delay(1000);
  digitalWrite(4, HIGH);
  delay(5000);
  digitalWrite(4, LOW);
  delay(2000);
}
```

Figura 14 – Código-fonte .ino gerado pela ferramenta proposta.

6. Resultados e discussões

No decorrer do trabalho surgiram alguns problemas quanto ao desenvolvimento da ferramenta, resultando na alteração do planejamento inicial do projeto. Um dos desafios mais importantes encontrados foi que, na geração de código-fonte para diferentes dispositivos, identificou-se a necessidade de incluir na interface da ferramenta um campo Classes de Dispositivos, no qual é exibido em um *Combo Box* as classes de dispositivos que acompanham a ferramenta. Assim com a escolha dessa classe, são listadas as ações relacionadas ao tipo de dispositivo.

Outro problema encontrado foi que no campo *setpoint* da interface da ferramenta, optou-se deixar por padrão o valor zero. O motivo deste campo não poder ficar nulo, é que quando o código-fonte é gerado, o *delay* para as ações dos dispositivos ficaria vazio, resultando em problemas de sintaxe no código gerado.

Tanto a geração de código-fonte como a extração de dados para o cenário de teste obtiveram êxito, pois o código foi gerado de forma correta, podendo ser carregado na IDE Arduino onde é executado o protótipo.

Outros modelos de cenários do Fritzing para extração de dados foram testados durante o desenvolvimento do trabalho, contendo dispositivos como: *LM35 Temperature Sensor*, *Light Dependent Resistor* (LDR) e *DC Motor.* Ambos dispositivos foram mapeados corretamente pela ferramenta. Entretanto, a extração de dados não se aplica para todos tipos de cenários pelo fato do arquivo XML exportado do Fritzing não apresentar um padrão de relacionamento entre as conexões.

7. Conclusão e trabalhos futuros

Este trabalho apresentou a proposta de uma ferramenta para a geração de código-fonte para a plataforma Arduino, em que modelos projetados na ferramenta Fritzing (a qual gera apenas a estrutura de um cenário) podem ganhar comportamento por meio da presente proposta.

A vantagem de se propor uma ferramenta com essa finalidade está, justamente, na possibilidade de se desenvolver aplicações para a plataforma de prototipagem Arduino sem que, necessariamente, o projetista que utiliza a ferramenta Fritzing conheça a linguagem de programação Arduino, a qual é necessária para que um cenário microcontrolado assuma um comportamento desejado.

Ademais, este trabalho pode servir de referência para projetos futuros dentro de um mesmo domínio de aplicação que envolva geração automática de código, criação de estrutura e comportamento de dispositivos e sistemas microcontrolados. Uma característica a ser melhorada é a capacidade de gerar código-fonte para diferentes dispositivos, buscando incluir um repositório de classes de dispositivos *online* para atender inúmeros cenários.

Referências

Arduino (2016). “Arduino”. Acesso em 21 de Março de 2016. Disponível em: https://www.arduino.cc/.

Atmel (2016). “Atmel Corporation”. Acesso em 17 de Março de 2016. Disponível em: http://www.atmel.com/.

Atmel (2016a). “Automação industrial”. Acesso em 18 de Março de 2016. Disponível em: http://www.atmel.com/pt/br/applications/industrialautomation/default.aspx.

Atmel (2016b). “Eletrônicos para dispositivos móveis”. Acesso em 18 de Março de 2016. Disponível em: http://www.atmel.com/pt/br/applications/mobile_electronics/default.aspx.

Castro, L. L. (2010). “Procedimentos de modelagem e uma ferramenta de geração automática de código”, Universidade Federal de Lavras, Minas Gerais – Brasil.

Corteletti, D. (2006). “Dossiê Técnico - Introdução à programação de microcontroladores Microchip PIC”, SENAI-RS, Centro Tecnológico de Mecatrônica.

Deitel, H. M., Deitel, P. J. (2005). “Java como programar”, 6º edição, Pearson Education – Br.

Embarcados (2016). “Arduino – Primeiros Passos”. Acesso em 27 de Fevereiro de 2016. Disponível em: http://www.embarcados.com.br/arduino-primeiros-passos/.

Fritzing (2016). “Fritzing”. Acesso em 26 de Fevereiro de 2016. Disponível em: http://fritzing.org/home/.

Gallassi, T. T., Martins L. E. G. (2011). “Um estudo exploratório sobre sistemas operacionais embarcados”, Fatec Americana, Curso Superior de Tecnologia em Análise de Sistemas e Tecnologia da Informação da Faculdade de Tecnologia de Americana – Fatec Americana, São Paulo - Brasil.

Indrusiak, L. S. (1996). “Linguagem Java”, Grupo JavaRS, JUG Rio Grande do Sul.

Oki, N., Mantovani S. C. A. (2013). “TEEEI - Projeto de Robôs Móveis”, Faculdade de Engenharia de Ilha Solteira - Departamento de Engenharia Elétrica, Ilha Solteira - Brasil.

Perozzo, R. F. (2011). “Framework para integração entre ambientes inteligentes e o sistema brasileiro de TV digital”, Universidade Federal do Rio Grande do Sul, Escola de Engenharia, Departamento de Engenharia Elétrica, Programa de Pós-Graduação em Engenharia Elétrica, Porto Alegre - Brasil.

Ramalho, J. C., Henriques, P. (2002). “XML & XSL da teoria à prática”, FCA - Editora Informática.

Retamal, A. M. (2008). “Feature-Driven Development”. Acesso em 19 de Março de 2016. Disponível em: http://www.heptagon.com.br/files/FDD-Processos.pdf.

Ribeiro, J., Novais, R. L., Mendes, T. S. (2005). “Geração Automática de Código Java a partir do Modelo de Dados”, Grupo de Informática Aplicada - Instituto Federal da Bahia – Campus Santo Amaro, Santo Amaro – BA – Brasil.

XPath (2016). “XML Path Language (XPath)”. Acesso em 02 de Agosto de 2016. Disponível em: https://www.w3.org/TR/xpath/.